UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructuras de Datos I

Taller de Testing - C++

Este taller contiene una introducción para familiarizarse con el sistema de testing. Primero se ofrecen ejercicios para generar casos de prueba de código existente, y luego se propone testear código propio. Todo lo que necesitan para el taller está en `template-alumnos.zip`.

No olvidarse de descomprimir la librería GTEST dentro del folder `lib`.

1. Corrida de casos de prueba

Correr los casos de test provistos para la función `bool esPrimo(int n)`, en el archivo *esPrimo.cpp*. La implementación es correcta?

Armar casos de test de caja blanca para la función `int puntaje(int n)`, en el archivo *puntaje.cpp*, pasando por todas las ramas (branches) del código.

2. Revisión de código existente

Justificar la elección de los casos de prueba en cada subítem. Además, decidir si la implementación provista en la carpeta **ej2** es correcta. Dada la adaptación del problema:
`int llenarTaxis(int g1, int g2, int g3)`[1]

Armar casos de test de caja negra según la partición de dominio que se encuentra al final del problema. Correr los tests sobre las diferentes implementaciones en la carpeta **ej2** y encontrar el error de cada una (no hace falta arreglarlas).

Enunciado del problema

A la salida de clases, un viernes a la noche, los estudiantes de Algo1 están invitados al cumpleaños de un compañero. Deciden ir en taxi, intentando optimizar el gasto total en el viaje. Los chicos están divididos en grupos de amigos de entre 1 y 3 personas y cada grupo de amigos quiere viajar en el mismo taxi. Las variables **g1**, **g2** y **g3**, indican la CANTIDAD de grupos con una, dos y tres personas, respectivamente. Cada taxi tiene capacidad para 4 personas cada uno y puede llevar a más de un grupo (por ejemplo, puede llevar dos grupos de dos personas: si **g2**=2).

Cuál es la mínima cantidad de taxis que necesitan para poder llegar todos?

La entrada son las tres variables: **g1**, **g2** y **g3**.
La salida es un número: la mínima cantidad de taxis que necesitan.

Partición del dominio

1. Todas las variables con el mismo valor,
2. Con grupos de diferentes tamaños:
   - *a*) g_2 par.
   - *b*) g_2 impar...
     1) ..., $g_1 > 0, g_3 > 0$ y $|g_1 - g_3| = 1$ ó 2
     2) ..., $g_1 > 0, g_3 > 0$ y $|g_1 - g_3| \neq 1$ ó 2

[1] http://codeforces.com/problemset/problem/158/B

3. Programación + testing

Implementar una solución de los siguientes problemas. Además, proveer casos de test de caja blanca y justificar su elección.

3.1. baldosasDelPiso

Trabajar sobre el problema `int baldosasDelPiso(int N, int M, int B)`[2]:

La facultad quiere cambiar las baldosas del piso de un aula rectangular (de dimensiones $M \times N$) sin huecos ni superposiciones. Las baldosas son cuadradas (de $B \times B$).

Las baldosas se pueden cortar para ponerlas en el piso, pero a lo sumo se puede usar un pedazo de cada una (es decir, no vale cortarlas a la mitad para tener dos más chicas).

Cuál es la mínima cantidad de baldosas que se necesita para cubrir el aula?

Partición del dominio

1. M y N divisibles por B.
2. Sólo M divisible por B.
3. Sólo N divisible por B.
4. Ninguno divisible por B.

Resolver los puntos:

1. Programar en C++ una solución al problema.
2. Crear un conjunto de casos de test de caja negra que contenga un caso por cada partición del dominio del problema mostrado arriba.
3. Crear un conjunto de casos de test de caja blanca que cubra todas las líneas del programa.
4. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas las ramas de decisiones (branches) del programa.

3.2. Padrón Electoral

Un estudiante de Algo1 está haciendo una pasantía en la Cámara Nacional Electoral. Su responsable le pide que haga un relevamiento de los electores a nivel nacional que votarán este 22 de octubre, indicando la cantidad de votantes que tienen obligación de hacerlo y para cuantos es opcional. Un rápido vistazo le permite comprobar a este chico que la base de datos de las fechas de nacimiento puede contener errores. Para encarar el problema, decide generar una función que reciba la fecha de nacimiento en tres variables enteras, verifique su validez, y devuelva una de las etiquetas siguientes:

- 0 = NO_VOTA
- 1 = OPCIONAL_MENOR
- 2 = OBLIGATORIO
- 3 = OPCIONAL_MAYOR
- 4 = ERROR

OPCIONAL_MENOR son aquellos votantes que tienen cumplidos 16 años el 22 de octubre y tienen menos de 18 años. OPCIONAL_MAYOR son aquellos votantes de 70 o más años al 22 de octubre.

Trabajar sobre la función `int validarVotante(int a, int m, int d)`: donde a es el año de nacimiento del registro, m el mes y d el dia.

Partición del dominio

1. Fechas inválidas.
2. Fechas válidas de votantes y no votantes.

[2] http://codeforces.com/problemset/problem/1/A

Resolver los puntos:

1. Programar en C++ una solución al problema.
2. Crear un conjunto de casos de test de caja negra que contenga un caso por cada partición del dominio del problema mostrado arriba.
3. Crear un conjunto de casos de test de caja blanca que cubra todas las líneas del programa.
4. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas las ramas de decisiones (branches) del programa.

3.3. contandoDragones (opcional)

Trabajar sobre la función:
`int contandoDragones(int T, int d1, int d2, int d3)`[3]:

Enunciado del problema

Ana Paula tiene insomnio, y eso la pone de muy mal humor. Desde hace un tiempo cuenta dragones para dormir porque las ovejas la aburrieron. Pero claro, también se aburrió de contar dragones. Entonces se empezó a imaginar que a algunos les hacía maldades. Por ejemplo, de un total de T dragones, a uno de cada d_1 dragones les tiró gas pimienta en la nariz, a uno de cada d_2 dragones los roció con un extintor, y a uno de cada d_3 les puso picante en la comida. Sabemos que $d_1 < d_2 < d_3$, y algún dragón puede sufrir más de una maldad.

¿Cuántos dragones no fueron víctimas de ninguna de sus maldades?

Pista: sale sin usar un ciclo que chequee los dragones uno por uno.

Partición del dominio

1. d_1, d_2 y d_3 coprimos. Se divide en:
   a) Coprimos de a pares
   b) No coprimos de a pares
2. d_1, d_2 y d_3 no coprimos. Se divide en:
   a) d_2 múltiplo de d_1 y d_3 múltiplo de d_2.
   b) No ocurre la condición de arriba.

Resolver los puntos:

1. Programar en C++ una solución al problema.
2. Crear un conjunto de casos de test de caja negra que contenga un caso por cada partición del dominio del problema mostrado arriba.
3. Crear un conjunto de casos de test de caja blanca que cubra todas las líneas del programa.
4. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas las ramas de decisiones (branches) del programa.

[3] http://codeforces.com/problemset/problem/148/A